CheatConsole - Documentation

This tool provides developers with a ready-to-use cheat console. It works as an interactive in-game “tablet” that displays a list of available commands. Each command is defined in code using an attribute (e.g. `[Cheat("alias", "description")]`) and, after recompilation, appears on the tablet screen. The console is delivered as a ready-made prefab (*CheatsTablet*) that can be placed directly into the scene.

Demo

The `CheatConsole_Demo` scene contains a ready-to-use and fully configured setup for testing the command tablet. Add examples to favorites and experiment freely. Some examples may produce errors (for example, commands that search for objects with a specific tag that does not exist). The main purpose of the demo is to showcase many examples, how they are defined, and how commands are displayed on the tablet.

PIN

The tablet is protected by a four-digit PIN stored in the `cheat-console-pin.txt` file (located in Resources). After entering the PIN, the command windows are loaded. The `CommandPin` component includes the `isUnlockedOnStart` field (default: *false*), which allows the PIN to be automatically accepted. This is a convenience feature for debugging. The PIN is stored for the lifetime of the application, so if the tablet is changed or recreated during a scene change, a previously entered PIN will be automatically accepted.

Navigation Menu

In the Unity top menu bar, an additional menu is available: `Tools/Cheat Console/`, which contains the **Auto Reload Enabled** toggle (when enabled, every script reload automatically refreshes the command list) and the **Force Update** button, which refreshes the command list manually when automatic reloading is disabled.

Implementation

To add this cheat console to your project, simply place the `CheatsTablet` prefab into the scene. It can be positioned anywhere. If the main camera is far away from the tablet, all of its Canvases are automatically hidden for performance reasons. This behavior is handled by the `AutoHideCanvas` script.

Adding Custom Commands

There are several conditions that must be met for a command to be correctly registered and executed by the system.

1. The attribute must be applied to a method in a class that inherits from `MonoBehaviour`.

2. Method parameters must be primitive types and/or commonly used Unity structures. If you need to support a custom type, refer to the `StringConverter` class to see how other known types are converted.

What the system supports:

1. Public and private methods;
2. Static and non-static methods;
3. `void` methods as well as methods that return any type (not tested with `async`, `Task`, or coroutines);
4. No limit on the number of parameters;
5. Commands for overloaded methods are allowed (in this case, a different command name must be provided);
6. Use of the `params` keyword is allowed;
7. *Extension methods* were not tested.

Available Command Parameters:

Parameter	Type	Default value	Description
`group`	`string`	`null`	Grouping allows filtering the commands displayed on the tablet by a specific group. This makes it possible to show different commands depending on the scene. Filtered groups are configured in the tablet inspector.
`commandName`	`strnig`	`null`	The literal command name that will be executed in the console. If `null` or empty, the method name is automatically converted from **PascalCase** to **kebab-case** (e.g. `MyCommandMethod` → `my-command-method`).
`commandStruct`	`string`	`null`	The formal command syntax containing placeholders for parameters. Example: `add <number> <text>` shows how the user should type the command.
`description`	`string`	`null`	A detailed description of the command behavior, effects, and any important notes or side effects before execution.
`examples`	`string[]`	empty (`new string[0]`)	An array of ready-to-use command examples with concrete parameter values. They can be executed directly to test the command.

Examples

The simplest command, without parameters.

```
[Cheat]
private static int SimpleCommand()
{
    return 0;
}
```

```
[Cheat(
    commandName: "sum-array",
    commandStruct: "sum-array <n1> <n2> ... <nN>",
    group: "group1",
    description: "Returns the sum of any number of integers.",
    /*examples:*/ "1 2 3", "10 20 30 40")]
private static void SumArray(params int[] values)
{
    int sum = 0;
    foreach (var v in values) sum += v;
    Debug.Log($"SumArray: sum = {sum}");
}
```

```
[Cheat(
    commandName: "echo-message",
    commandStruct: "echo-message <value> <message>",
    description: "Multiplies the float value by 2 and then appends a
message.",
    "3.5 Hello", "10.0 World")]
private static void EchoMessage(float value, string message)
{
    Debug.Log($"EchoMessage: {(value * 2):0.##} -> {message}");
}
```

Supported Command Parameter Types

The system supports:

- all **primitive types** (`int` , `float` , `double` , `decimal` , `bool` , `string` , etc.),
- all **enumeration types (** `enum` **)**,
- and the following Unity types:

Unity Type	Console Format
`Vector2`	`x,y`
`Vector2Int`	`x,y`

Unity Type	Console Format
`Vector3`	`x,y,z`
`Vector3Int`	`x,y,z`
`Vector4`	`x,y,z,w`
`Quaternion`	`x,y,z,w`
`Color`	`r,g,b,a` *(values 0–1)*
`Color32`	`r,g,b,a` *(values 0–255)*
`Rect`	`x,y,width,height`
`Bounds`	`centerX,centerY,centerZ,sizeX,sizeY,sizeZ`
`RectInt`	`x,y,width,height`
`BoundsInt`	`centerX,centerY,centerZ,sizeX,sizeY,sizeZ`
`Ray`	`originX,originY,originZ,directionX,directionY,directionZ`
`Ray2D`	`originX,originY,directionX,directionY`
`Plane`	`normalX,normalY,normalZ,distance`
`Matrix4x4`	16 comma-separated values, in row-major order

> 📌 For floating-point numbers (`float` , `double` , `decimal`), a dot (`.`) entered in the console is automatically converted to a comma (`,`), avoiding localization issues in Unity.

Tablet Features

The tablet is the main user interface of the console system. It allows browsing and executing commands without manually typing them. It consists of three sections:

All Commands List

Contains the complete list of commands detected in the project (based on methods marked with the `[Cheat]` attribute in classes inheriting from `MonoBehaviour`). Clicking an entry opens the details window for the selected command.

Favorites List

Displays commands (more precisely, specific command examples) that have been added to favorites by the user. This allows instant execution of a saved command example and quick access to the command details window to view additional information and other examples.

Command Details Window

Displays full information about the selected command:

- Command name – as defined by the attribute or automatically generated from the method name.
- Description – the purpose and behavior of the command.

- List of all examples – ready-to-run argument sets that can be executed with a single click.

From this window, you can:

- Execute any example,
- Add an example to the favorites list.

Requirements and Dependencies

- Unity (recommended version 6.0 or newer)

Asset Store

- Lean Pool
- Prime Tween

Packages

- Localization (with locale: `pl-PL` or `en`)